Available online at www.sciencedirect.com
SCIENCE DIRECT®

Discrete Applied Mathematics 145 (2005) 198–209

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Linear-time modular decomposition of directed graphs

Ross M. McConnell[a], Fabien de Montgolfier[b]

[a]*Colorado State University, Fort Collins, CO 80523-1873, USA*
[b]*LIAFA, Université Paris 2, 2 place Jussieu, 75005 Paris, France*

Received 18 February 2002; received in revised form 16 April 2003; accepted 26 February 2004
Available online 2 October 2004

Abstract

Modular decomposition of graphs is a powerful tool with many applications in graph theory and optimization. There are efficient linear-time algorithms that compute the decomposition for undirected graphs. The best previously published time bound for directed graphs is $O(n + m \log n)$, where n is the number of vertices and m is the number of edges. We give an $O(n + m)$-time algorithm.

Keywords: Modular decomposition; Algorithm; Tournaments; Directed graphs; Partitive families; Tree-decomposable families; 2-structures

1. Introduction

A *module* in a graph $G = (V, E)$ is a set X of vertices such that each vertex in $V \setminus X$ has a uniform relationship to all members of X. That is, if $y \in V \setminus X$, then y has directed edges to all members of X or to none of them, and all members of X have directed edges to y or none of them do. (See Fig. 1.) Different members y and y' of $V \setminus X$ can have different relationships to members of X. However, for instance, y can have directed edges to all members of X when y' has directed edges to none of them. The members of X can have arbitrary relationships to each other, as can the members of $V \setminus X$.

It is not hard to see that if X and Y are two disjoint modules, then if some vertex of Y is a neighbor of some vertex of X, then all vertices of Y are neighbors of all vertices of X. Therefore, Y can be considered unambiguously to be a *neighbor* of X or a *non-neighbor* of X. If $\mathscr{P}$ is a nontrivial partition of V such that each member of $\mathscr{P}$ is a module, this observation gives rise to a *quotient graph*, which is the graph of adjacencies between members of $\mathscr{P}$. (See Fig. 2.) The subgraphs induced by the members of $\mathscr{P}$ record the relationships in G that are not captured by the quotient. Together, the quotient and factors give a representation of G.

Further simplification can often be obtained by decomposing the factors and the quotient recursively. The *modular decomposition* is a unique, canonical way to do this that implicitly represents all possible ways the decompose the graph into quotients and factors. It can be represented by a rooted tree.

Modular decomposition theory originates from Gallai's work about transitive orientation [13]. Möhring and Radermacher [22,23] survey the topic.

The class of *cographs* (and some extensions like P_4-sparse, P_4-reducible, and P_4-tidy) [5,15,17,18] are classes where a graph is uniquely defined by the properties of its modular decomposition. A great number of NP-hard optimization problems for graphs can be easily solved if a solution is known for every quotient graph in the modular decomposition. If every quotient is small, this gives an efficient solution. Its famous applications include transitive orientation [13], weighted maximum clique, and coloring.

E-mail address: rmm@cs.colostate.edu (R.M. McConnell), fm@liafa.jussieu.fr (F. de Montgolfier).

0166-218X/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2004.02.017

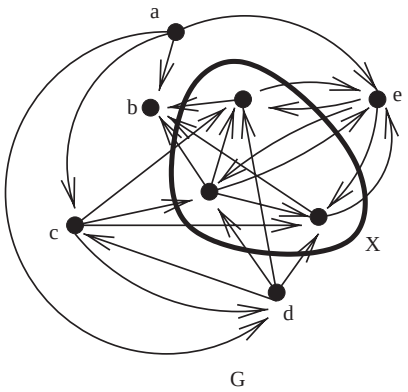

Fig. 1. A module of an undirected graph is a set X of vertices such that each vertex $y \in V \setminus X$ has a uniform relationship to all members of X. For instance, every member of X is a neighbor of d and no member of X has d as a neighbor, so they all have the same relationship to d.

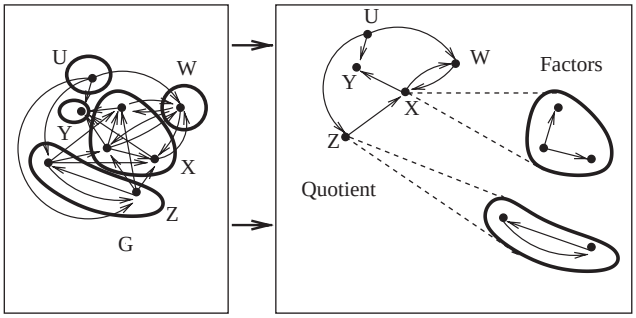

Fig. 2. If A and B are disjoint modules, then if B contains a neighbor of a vertex in A, every member of B is a neighbor of every member of A. In this case, B is *adjacent* to A. A partition $\mathscr{P}$ of G where each member of $\mathscr{P}$ is a module defines a *quotient graph*, which describes the adjacencies among members of $\mathscr{P}$. The quotient, together with the subgraphs induced by the nontrivial members of $\mathscr{P}$, give a representation of G, since the edges of G can be recovered from them.

Modular decomposition is also used in graph drawing. Many classes, such as interval graphs or permutation graphs have simple recognition algorithms using modular decomposition (see [1,14] for a survey). Fewer directed graph classes are known, but modular decomposition can help in their recognition (see [24] for instance).

Some width parameters are also closely related to the modular decomposition. The clique-width of a graph is the maximum of clique-widths of quotient graphs in the modular decomposition tree. Classes with a finite number of possible quotients therefore have a bounded clique-width (2 for cographs, 3 for P_4-sparse, P_4-reducible, and P_4-tidy).

Many algorithms of various complexities have appeared, beginning in the 1960s. The first linear-time algorithm was given by [20,21], which was quickly followed by other linear time algorithms that use quite different approaches to the problem [6,8]. The best previous time bound for the directed case is $O(n^2)$ [12,19] or $O(m \log n)$[9]. Here we present an $O(n + m)$-time algorithm.

A graph G can be thought of as a coloring of the edges of the complete graph with two colors, one corresponding to edges that are contained in G and one corresponding to edges that are in its complement. This abstraction awards no special status to G over its complement; the modules are the same on both graphs. The *2-structures* are a generalization of graphs: a 2-structure is a coloring of the complete digraph with k color, instead of two. This object was introduced by Ehrenfeucht and Rozenberg [10,11], who gave a generalization of modular decomposition to 2-structures.

Chein et al. [3] characterized the properties that a family of sets, such as the modules of a graph, must have in order to have a decomposition tree such as the modular decomposition. Such families are called *partitive set families*. Our algorithm exploits the modular decomposition of a 2-structure, as well as the fact that the intersection of two partitive set families is a partitive set family. We develop a procedure for finding the tree decomposition of the intersection, given the tree decompositions of the two families.

The next section gives basic definitions and concepts. Section 3 presents the algorithm for finding the decomposition tree of the intersection of two partitive families. The fourth section gives a novel and simple algorithm for decomposition of tournaments that we use in the main algorithm. The fifth part of the paper gives the directed graph decomposition algorithm itself.

2. Preliminaries

2.1. Graphs and digraphs

Let $G=(V, E)$ be a finite *directed graph* (or simply *digraph*) with vertex-set $V=V(G)$ and arc-set $E=E(G) \subseteq V(G)\times V(G)$. Here a digraph is loopless ($(u, u) \notin E$). The digraph *induced* by $X \subseteq V$ is $G[X] = (X, E \cap (X \times X))$. The pair (u, v) a *simple arc* if $(u, v) \in E$ and $(v, u) \notin E$, an *edge* if $(u, v) \in E$ and $(v, u) \in E$, and a *non-edge* if $(u, v) \notin E$ and $(v, u) \notin E$. Let $n(G)$ denote the number of vertices of G and $m(G)$ denote the number of arcs (with edges being counted twice). Let n and m denote these when G is understood.

Let $N^+(v) = \{u|(v, u) \in E\}$ and let $N^-(v) = \{u|(u, v) \in E\}$. If X is a module, we may write $N^+(X)$ and $N^-(X)$. A digraph can be stored in $O(n + m)$ space using *adjacency-list* representation [4]. A digraph is *connected* if there is no partition of V in two non-empty sets with no arcs between them; a maximal connected subgraph is a *component*. An *undirected graph* (or simply *graph*) has no simple arc. A *tree* is a connected directed graph such that every vertex except one (the *root*) is the origin of one simple arc. A *stable set* is a digraph such that $E = \emptyset$, a *clique* (or complete digraph) is a digraph, such that $E = \{(u, v)|u \neq v\}$, and a *tournament* is a digraph where there is a simple arc between each pair of vertices. For our purposes, a *linear order* is an acyclic tournament. A linear order has a unique topological sort.

2.2. Partitive families

The *symmetric difference* of two sets is $A\Delta B = (A \cup B)\backslash(A \cap B)$. Two sets X and Y *overlap* if they intersect, but neither is a subset of the other. That is, they overlap if $X\backslash Y$, $X \cap Y$, and $Y\backslash X$ are all nonempty.

Let V be a finite set and $\mathscr{F}$ a family (set) of subsets of V. Let $Size(\mathscr{F}) = \sum_{F\in\mathscr{F}} |F|$. $\mathscr{F}$ is *tree-like* if $\emptyset \notin \mathscr{F}$, $V \in \mathscr{F}$, $\{x\} \in \mathscr{F}$ for all $x \in V$, and for all $X, Y \in \mathscr{F}$, X and Y do not overlap.

Lemma 1. *The Hasse diagram* (*digraph of the transitive reduction*) *of the subset relation on a tree-like family is a rooted tree.*

Let us call the Hasse diagram of such a family the family's *inclusion tree*. This defines a parent relation on members of $\mathscr{F}$, and allows us to speak of the *siblings* and *children* of a member of $\mathscr{F}$.

The following is well-known:

Lemma 2. *If $\mathscr{F}$ is a tree-like family on domain V and X is a nonempty subset of V that does not overlap any member of $\mathscr{F}$, then X is a union of one or more siblings in $\mathscr{F}$'s inclusion tree.*

Proof. Let Y be the least common ancestor of X. If X is not a union of siblings, then X fails to contain some child A of Y that it intersects. Then X overlaps A, a contradiction. □

$\mathscr{F}$ is a *strongly partitive family* [3] (also called *decomposable set family* by [22]) if:

- $V \in \mathscr{F}$, $\emptyset \notin \mathscr{F}$, and $\forall v \in V$, $\{v\} \in \mathscr{F}$.
- $\forall X, Y \in \mathscr{F}$, if X and Y overlap, then $X \cap Y \in \mathscr{F}$, $X \cup Y \in \mathscr{F}$ and $X\Delta Y \in \mathscr{F}$.

In this paper we assume that the empty set is not a member of $\mathscr{F}$. A member of a partitive family $\mathscr{F}$ is said to be *strong* if no other member of $\mathscr{F}$ overlaps it, otherwise it is weak. $\mathscr{S}(\mathscr{F})$ is the family of strong sets of $\mathscr{F}$. Though $\mathscr{F}$ is not a tree-like family, $S(\mathscr{F})$ is. Let $T(\mathscr{F})$ denote the inclusion tree of $\mathscr{S}(\mathscr{F})$.

Theorem 3 (*Chein et al. [3], Möhring [22]*). *Let $\mathscr{F}$ be a strongly partitive family and let X be an internal node of $T(\mathscr{F})$ with children $S_1, S_2, \ldots, S_k$. Then X is of one of the following two types*:

- *Complete*: *For every $I \subset \{1, \ldots, k\}$, such that $1 < |I| < k$, $\bigcup_{i\in I} S_i \in \mathscr{F}$*
- *Prime*: *For every $I \subset \{1, \ldots, k\}$, such that $1 < |I| < k$, $\bigcup_{i\in I} S_i \notin \mathscr{F}$*

By Lemma 2, this implies that a set is a member of $\mathscr{F}$ iff it is a node of $T(\mathscr{F})$ or a union of children of a complete node in $T(\mathscr{F})$ (as a module cannot overlap a strong module).

Notice that $Size(\mathscr{F})$ can be exponential in $|V|$ (the boolean family 2^V is strongly partitive) but that $Size(S(\mathscr{F})) \leqslant |V|^2$. $T(\mathscr{F})$ is thus a polynomial-size representation of the family.

$\mathscr{F}$ is a *weakly partitive family* if:

- $V \in \mathscr{F}$, $\emptyset \notin \mathscr{F}$, and $\forall v \in V$, $\{v\} \in \mathscr{F}$
- $\forall X, Y \in \mathscr{F}$, if X and Y overlap, then $X \cap Y \in \mathscr{F}$, $X \cup Y \in \mathscr{F}$, $X \backslash Y \in \mathscr{F}$, and $Y \backslash X \in \mathscr{F}$.

When X and Y are overlapping members of a strongly partitive family, then so is $X \Delta Y$, and this member overlaps X. Therefore, $X \backslash Y = X \cap (X \Delta Y)$ is also a member of the family. Similarly, $Y \backslash X$ is in the family. This implies that a strongly partitive family is a weakly partitive family, but the converse is not true.

Theorem 4 (*Habib [16], Möhring and Radermacher [23]*). *Let $\mathscr{F}$ be a weakly partitive family, let X be an internal node of $T(\mathscr{F})$, and let $S_1, S_2, \ldots, S_k$ be the children of X. Then X is of one of the following three types*:

- *Complete*: *For every $I \subset \{1, \ldots, k\}$, such that $1 < |I| < k$, $\bigcup_{i \in I} S_i \in \mathscr{F}$*
- *Prime*: *For every $I \subset \{1, \ldots, k\}$, such that $1 < |I| < k$, $\bigcup_{i \in I} S_i \notin \mathscr{F}$*
- *Linear*: *There exists an ordering of $\{1, 2, \ldots, k\}$ such that if $I \subset \{1, \ldots, k\}$ and $1 < |I| < k$, then $\bigcup_{i \in I} S_i \in \mathscr{F}$ iff the members of I are consecutive in the ordering.*

Conversely, by Lemma 2, if $\mathscr{F}$ is a weak partitive family, $Y \subseteq V$ is a member of $\mathscr{F}$ iff it is either a node of $T(\mathscr{F})$, the union of a set of children of a complete node of $T(\mathscr{F})$, or the union of a consecutive set of children in the ordering of a linear node.

2.3. 2-structures

A 2-*structure* [10] is a triple $G = (V, E, k)$, where V is a finite vertex-set, $k \in \mathbb{N}$, and $E : V \times V \mapsto \{1, \ldots, k\}$ is a *coloring* function. A 2-structure is *symmetric* if $E(x, y) = E(y, x)$. Notice that for $k = 2$ a 2-structure is a digraph, and a symmetric 2-structure is a graph when one of the color classes is interpreted as the edges and the other as the non-edges. Furthermore, a loopless multigraph $G = (V, E)$ where E is a multiset of pairs of vertices may be seen as a 2-structure $G = (V, E', k)$, where E' counts the number of edges between two vertices and k is the maximum of E'. $M \subseteq V$ is a *module* of a 2-structure (V, E, k) if it is nonempty and

$$\forall x, y \in M \ \forall z \notin M \ \ E(x, z) = E(y, z) \text{ and } E(z, x) = E(z, y)$$

In other words, a module is a 2-structure is a set X of vertices that have a uniform relationship to each $z \in V \backslash X$. The *trivial modules* are V and its one-element (singleton) subsets.

Theorem 5 (*Ehrenfeucht and G. Rozenberg [11]*).

- *The modules of a 2-structure form a weakly partitive family.*
- *The modules of a symmetric 2-structure form a strongly partitive family.*

The modular decomposition of a 2-structure H is the tree $T(H)$ given by Theorems 5 and 4 or Theorem 3, depending on whether H is symmetric.

If X is a nonempty subset of V, and H is a 2-structure, let $H[X]$ denote the substructure induced by X, that is, X and the coloring of $X \times X$ given by H.

If X and Y are disjoint modules of H, then all members of $X \times Y$ are colored with the same color, and all members of $Y \times X$ are colored with the same color. If $\mathscr{P}$ is a partition of V where every partition class is a module, the *quotient* induced by $\mathscr{P}$ is the 2-structure with the members of $\mathscr{P}$ as vertices, and where for $X, Y \in \mathscr{P}$, the color of (X, Y) is the color of the edges of $X \times Y$ in H.

Let M be a node of $T(H)$ and let $M_1 \ldots M_k$ be its children. Since $\{M_1, M_2, \ldots, M_k\}$ is a partition of the vertices of $H[M]$ where every part is a module, it defines a quotient on $H[M]$. Let us call this *Ms quotient* in $T(H)$.

A 2-structure is *prime* if it has only trivial modules. It is a *c-clique* if $E(x, y) = c$ for all x and y. It is a (c, c')-order if $E(x, y) \in \{c, c'\}$ for all x and y, and the relation $x \mathscr{R} y$ iff $E(x, y) = c$ is a total order.

Proposition 6 (*Ehrenfeucht and G. Rozenberg [11]*). *Let M be a strong module of a 2-structure G.*

- *If M is prime* (*in the sense of Theorem* 4), *the quotient of M is a prime 2-structure.*
- *If M is complete, there exists c such that the quotient of M is a c-clique.*
- *If M is linear there exists c and c' such that the quotient of M is a (c, c')-order.*

Let us say that a node is *c-complete* if its quotient is a *c*-clique, and (c, c')-*linear* if its quotient is a (c, c')-order.

2.4. Modular decomposition of digraphs

The modules of a digraph are obtained by treating it as a 2-structure on V with two colors, one for edges and one for non-edges. The properties of modules apply to graphs as a special case. By Proposition 6, if M is a linear node of $T(G)$, then its quotient is a total order, and if it is a complete node of $T(G)$, then its quotient is a clique or a stable set. A complete node is a *series node* if its quotient is a clique and a *parallel node* if its quotient is a stable set.

Notice that a digraph has at most $2n-1$ strong modules (as they form an inclusion tree with n leaves), while there can be 2^n different modules in a digraph (*e.g.* a stable set).

A vertex v *cuts* a set $S \subset V$ if $v \notin S$ and S is not a module of $G[S \cup \{v\}]$. The vertices that cut S are its *cutter-set*. M is a module if and only if its cutter-set is empty.

3. Intersection of strongly partitive set families

Let V be a set and $\mathscr{F}_a$, $\mathscr{F}_b$ be two partitive families on V. The *intersection* of $\mathscr{F}_a$ and $\mathscr{F}_b$ is $\mathscr{F} = \mathscr{F}_a \cap \mathscr{F}_b$, the family of sets that are members of both families.

Lemma 7. *The intersection of two strongly partitive families is a strongly partitive family.*

Proof. Let $\mathscr{F}_a$ and $\mathscr{F}_b$ be the two families. If X and Y are overlapping members of $\mathscr{F}_a \cap \mathscr{F}_b$, they are members of $\mathscr{F}_a$, so $X \cup Y$, $X \cap Y$, and $X \Delta Y$ are members of $\mathscr{F}_a$. The same is true of $\mathscr{F}_b$, so $X \cup Y$, $X \cap Y$, and $X \Delta Y$ are members of $\mathscr{F}_a \cap \mathscr{F}_b$. □

This suggests a binary operator on decomposition trees over domain V. Given two decomposition trees T_a and T_b of strongly partitive families on domain V, let $T_a \wedge T_b$ denote the partitive tree of $\mathscr{F}(T_a) \cap \mathscr{F}(T_b)$, which exists by Lemma 7.

In this section, we give an algorithm for computing $T_a \wedge T_b$ efficiently, given partitive trees T_a and T_b on the same domain V. In the rest of this section, let $\mathscr{F}_a$ and $\mathscr{F}_b$ denote the partitive families represented by T_a and T_b, and let $\mathscr{I} = \mathscr{F}_a \cap \mathscr{F}_b$.

Given a partitive tree T, let $\mathscr{F}(T)$ be the partitive set family that it represents. Let $\mathscr{S}(T)$ denote the strong members of that family. That is, $\mathscr{S}(T)$ is just the set of nodes of T.

$P_a(S)$ is the smallest node of T_a that contains S as a proper subset. Notice that if S is a union of siblings in T_a then $P_a(S)$ is their parent. Let $P_b(S)$ be defined in the same way on T_b.

Given an arbitrary set family $\mathscr{S}$ of subsets of domain V, let the *overlap graph* $\mathrm{O}(\mathscr{S})$ denote the graph whose vertices are the members of $\mathscr{S}$ and whose edges are the pairs $\{(A, B) | A \text{ and } B \text{ are overlapping members of } \mathscr{S}\}$. The connected components of $\mathrm{O}(\mathscr{S})$ are known as the *overlap components*.

Lemma 8. *If $\mathscr{C}$ is an overlap component of $\mathscr{S}$ and X is a set that overlaps $\bigcup \mathscr{C}$, then X overlaps some $S \in \mathscr{C}$.*

Proof. Since X overlaps $\bigcup \mathscr{C}$, it is not contained in any member of $\mathscr{C}$. Suppose X overlaps no member of $\mathscr{C}$. Let $\{\mathscr{A}, \mathscr{B}\}$ be the partition of members of $\mathscr{C}$ that are disjoint from X and contained in X, respectively. Since X overlaps $\bigcup \mathscr{C}$, each of $\mathscr{A}$ and $\mathscr{B}$ is nonempty. No member of $\mathscr{A}$ overlaps any member of $\mathscr{B}$, contradicting the assumption that $\mathscr{C}$ is an overlap component. □

Given strongly partitive trees T_a and T_b, let $\mathscr{O}(T_a, T_b) = \{\bigcup \mathscr{C} | \mathscr{C} \text{ is an overlap component of } \mathscr{S}(T_a) \cup \mathscr{S}(T_b)\}$. V and its singleton subsets are members of $\mathscr{O}(T_a, T_b)$. If $\bigcup \mathscr{C}$ overlaps $\bigcup \mathscr{D}$, then $\bigcup \mathscr{C}$ overlaps a member of $\mathscr{D}$, according to Lemma 8, and so $\mathscr{C}$ and $\mathscr{D}$ are the same overlap component. Therfore no members of $\mathscr{O}(T_a, T_b)$ overlap: $\mathscr{O}(T_a, T_b)$ is a tree-like family. According to Lemma 8 a node X of T_a or of T_b cannot overlap $\bigcup \mathscr{C}$, or else it would be in that component, therefore no nodes of T_a or T_b overlap a member of $\mathscr{O}(T_a, T_b)$.

No member of $\mathscr{I}$ overlaps a node of T_a or of T_b, so by Lemma 8, no member of $\mathscr{I}$ overlaps any member $\mathscr{O}(T_a, T_b)$. This gives the following by Lemma 2:

Lemma 9. *Every node of T_a and T_b, and every member of $\mathscr{I}$, is a union of siblings in inclusion tree of $\mathscr{O}(T_a, T_b)$.*

Let $\mathscr{A} = \{S | S \in \mathscr{O}(T_a, T_b)$ and S is a node of T_a or $P_a(S)$ is not prime in $T_a\}$. Let $\mathscr{B}$ be defined analogously on T_b. If T_a and T_b are strongly partitive trees, then let $\mathscr{U}(T_a, T_b) = \mathscr{A} \cap \mathscr{B}$.

Let R_U be the following relation on members of $\mathscr{U}(T_a, T_b)$: for $X, Y \in \mathscr{U}(T_a, T_b)$, $X R_U Y$ iff $P_a(X) = P_a(Y)$, $P_b(X) = P_b(Y)$, and neither of these nodes is prime.

Clearly, R_U is an equivalence relation. Let $\mathscr{S}(T_a, T_b) = \mathscr{U}(T_a, T_b) \cup \{\bigcup \mathscr{D} | \mathscr{D}$ is an equivalence class in $R_U\}$.

Theorem 10. *If T_a and T_b are the decomposition trees of strongly partitive families, then $\mathscr{S}(T_a, T_b)$ is the set of nodes of $T(\mathscr{I})$.*

Proof. Suppose $X \in \mathscr{O}(T_a, T_b)$ is a member of $\mathscr{I}$. Then it must be a node of T_a or a union of children of a complete node in T_a, hence, by Lemma 9 it is a member of $\mathscr{A}$. Similarly, it is a member of $\mathscr{B}$, so it is a member of $\mathscr{A} \cap \mathscr{B}$. X overlaps with no member of $\mathscr{I}$, hence it is a node of $T_{\mathscr{I}}$.

If $X \in \mathscr{I}$ is a strong member of $\mathscr{F}_a$, then it is a node of T_a, and, as it is also a member of $\mathscr{F}_b$, no node of T_b overlap it (in $\mathscr{F}_b$, M is either strong or the union of strong siblings), therefore X is the sole member of its equivalence class. The union of its overlap component is exactly M, and therefore $M \in \mathscr{O}(T_a, T_b)$. If M is a strong member of $\mathscr{F}_b$ then also $X \in \mathscr{O}(T_a, T_b)$. Any member Z of $\mathscr{I}$ that is not a member of $\mathscr{O}(T_a, T_b)$ is therefore weak in both $\mathscr{F}_a$ and $\mathscr{F}_b$. Z is then the union of sibling strong sets of $\mathscr{F}_a$, sons of a node Y_a of T_a, and the union of sibling strong sets of $\mathscr{F}_b$, sons of a node Y_b of T_b. Z is therefore the union of some subfamily of the equivalence class of R_Y corresponding to Y_a and Y_b. Every union of members of this equivalence class is a member of $\mathscr{I}$, so if Z is a node of $T(\mathscr{I})$, it must be the union of the entire equivalence class to avoid overlapping other such unions. This is also sufficient: if Z is the union of the entire equivalence class, no member of $\mathscr{I}$ overlaps Y_a or Y_b, and therefore no member of $\mathscr{I}$ overlaps Z, hence Z is a node of $T(\mathscr{I})$. □

We now describe some basic algorithmic tools.

Lemma 11. *Given a tree-like family $\mathscr{F}$, it takes* $O(Size(\mathscr{F}))$ *time to construct its inclusion tree.*

Proof. If V is the domain, $Size(\mathscr{F}) = \Omega(|V|)$, since $V \in \mathscr{F}$. Sort the members of $\mathscr{F}$ by size. This takes $O(Size(\mathscr{F}))$ time when using bucket sort, since the sizes are in the ranges from 1 to $|V|$ [4]. Then, create a list, for each $x \in V$, of the members of $\mathscr{F}$ that contain x, in ascending order of size. This can be accomplished by visiting each $Y \in \mathscr{F}$ in descending order of size, and for each $x \in Y$, inserting a pointer to Y to the front of xs list. This takes $O(Size(\mathscr{F}))$ time. Then, visit each member x of V, putting a parent pointer from each member of xs list to its successor in xs list if there isn't already one, as these are the chain of ancestors of $\{x\}$. □

Next, consider Algorithm 1 which is given in [26], and which we reproduce here for completeness. Given an inclusion tree on a domain V and an arbitrary $X \subseteq V$, it finds the maximal members of $\mathscr{F}$ that are subsets of X. The algorithm runs in $O(|X|)$ time, with linear-time initializations performed once.

```
Input: An inclusion tree T on V and an arbitrary subset X of V
Output: The maximal nodes N_1, ... N_k of T such that N_i ⊂ X
begin
    foreach leaf N = {x} of T, x ∈ X do
        mark N
    end
    foreach node N of T such that all its children S_1, ... S_k are marked do
        Unmark S_1, ... S_k
        mark N
    end
end
```

Algorithm 1. Mark the maximal nodes of an inclusion tree that are subsets of $X \subseteq V$

Let an inclusion tree be *initialized* if each node carries a parent pointer, a list of pointers to its children, an initialized field for marking, a record of how many children it has, and an initialized field for recording how many of its children are marked. The next lemma follows easily from Algorithm 1.

Lemma 12 (*Spinrad [26]*). *Given an initialized inclusion tree of a tree-like set family $\mathscr{F}$ on domain V and a set $X \subseteq V$, it takes* $O(|X|)$ *time to find the maximal members of $\mathscr{F}$ that are subsets of X, and then reinitialize the tree.*

Corollary 13. *Given the decomposition tree of a strongly partitive family $\mathscr{F}$ on domain V and $X \subseteq V$, it takes* $O(|X|)$ *time to determine whether* $X \in \mathscr{F}$.

Proof. X is a member of $\mathscr{F}$ iff it is a node of the decomposition tree or a union of children of a complete node. By Lemma 12, we may find the maximal nodes of the decomposition tree that are subsets of X, and verify that if there is more than one of them, they share a complete parent. □

Theorem 14 (*Dahlhaus [7]*). *Given a set family $\mathscr{S}$ on domain V, it takes* $O(|V| + Size(\mathscr{S}))$ *time to find the overlap components of* $\mathscr{S}$.

The algorithm, which is straightforward to implement, finds the components without actually computing the overlap graph. Finding the union of each of these components using an initialized boolean array of size $|V|$ gives $\mathscr{C}(T_a, T_b)$.

The following is the main result of this section.

Theorem 15. *Given decomposition trees T_a and T_b of strongly partitive families on domain V, it takes* $O(Size(S(\mathscr{F}_a)) + Size(S(\mathscr{F}_b)))$ *time to find* $T(\mathscr{I}) = T_a \wedge T_b$.

Proof. By Theorem 14, the bound is observed for finding the overlap components of the nodes $\mathscr{S}(\mathscr{F}_a) \cup \mathscr{S}(\mathscr{F}_b)$ of T_a and T_b. The union of each component is easily found within the bound using a boolean array of size $O(|V|)$ that is initialized once, and left in an initialized state after each union operation. This gives $\mathscr{C}(T_a, T_b)$. By Lemma 11, we may find the inclusion tree of $\mathscr{C}(T_a, T_b)$ within the bound.

By Corollary 13, Algorithm 1 can be used to test each node in this tree for membership in $\mathscr{A}$ and for membership in $\mathscr{B}$ in time linear in $Size(\mathscr{C}(T_a, T_b))$, which gives $\mathscr{U}(T_a, T_b)$.

We may then number the members of $\mathscr{U}(T_a, T_b)$ from 1 to $O(n)$. Algorithm 1 can then be used to find $P_a(X)$ and $P_b(X)$ for each $X \in \mathscr{U}(T_a, T_b)$ in time $O(Size(\mathscr{U}(T_a, T_b))$. The number labels of $P_a(X)$ and $P_b(X)$ give a pair of integers, each from 1 to $O(n)$; bucket sorting the members of $\mathscr{U}(T_a, T_b)$ according to this number pair gives the equivalence classes of R_U. The union of each equivalence class can be found in linear time using an initialized boolean array of size $O(|V|)$. □

In this paper we use the following application of this theorem. Let H and H' be two symmetric 2-structures on domain V. Let $H \wedge H'$, denote the 2-structure on domain V where two arcs have the same label iff they have the same label in both H and H'.

Lemma 16. *If H and H' are symmetric 2-structures on domain V, then*

$$T(H \wedge H') = T(H) \wedge T(H').$$

Proof. Let $\mathscr{F}$ be the modules of H and $\mathscr{F}'$ be the modules of H'. By the definitions, $M \subseteq V$ is a module of both H and H' iff for $x \in V \backslash M$, all edges between x and M have the same color in $H \wedge H'$. Therefore, the modules of $H \wedge H'$ are $\mathscr{F} \cap \mathscr{F}'$, whose decomposition tree is $T(H) \wedge T(H')$. □

4. Modular decomposition of tournaments

A *factorizing permutation* σ of a graph G is a linear ordering of $V(G)$ such that every strong module of G is a factor (interval) of σ. An embedding of $T(G)$ gives a factorizing permutation, just by reading its leaves from left to right. Conversely, Capelle, Habib and Montgolfier [2] give an $O(n+m)$ algorithm for retrieving $T(G)$, given a factorizing permutation. The algorithm that we give in this section for finding the modular decomposition of a tournament does so by computing a factorizing permutation and then making use of this result.

Input: A tournament $G = (V, E)$
Output: A perfect factorizing permutation $\mathcal{P}_n$ of G
begin
  Let $v_1 \ldots v_n$ be any ordering of V
  $\mathcal{P}_0 \leftarrow \{V\}$
  for *i from 1 to n* **do**
    Let C be the class of $\mathcal{P}_{i-1}$ such that $v_i \in C$
    $\mathcal{P}_i \leftarrow \mathcal{P}_{i-1}$ where C is replaced by $C \cap N^-(v_i)$; $\{v_i\}$; $C \cap N^+(v_i)$ consecutively
  end
end

Algorithm 2. A perfect factorizing permutation $\mathcal{P}_n$ of G

Let us say that a factorizing permutation is *perfect* if all modules of G, not just the strong modules, are intervals in the ordering.

A factorizing permutation exists for every graph, but a requirement for a perfect factorizing permutation to exist is that all nodes of the modular decomposition be prime or linear. All tournaments admit a perfect factorizing permutation. The factorizing permutation computed by the algorithm of this section is a perfect one, a fact that we make use of in a later section.

The algorithm uses *ordered partition refinement* algorithm [25]. An *ordered partition* is a list $\mathscr{P}$ of non-empty and pairwise-disjoint subsets (*classes*) of a set V, whose union is V, with a total order on the classes.

At each step, up to three new classes are substituted for an old one. Empty classes are not inserted.

Correctness of the algorithm: The correctness of this algorithm is a consequence of the following three invariants:

Invariant 1. *For all $0 \leqslant i \leqslant n$, $\mathscr{P}_i$ is an ordered partition of V having at least i singleton classes.*

Invariant 2. *Let C be a class of $\mathscr{P}_i$ having more than one vertex.*

- *If C is not the leftmost class of $\mathscr{P}_i$, then the class on the left of C is a singleton class $\{v_j\}$, $j \leqslant i$. Furthermore, $C \subset N^+(v_j)$.*
- *If C is not the rightmost class of $\mathscr{P}_i$, then the class on the right of C is a singleton class $\{v_k\}$, $k \leqslant i$. Furthermore, $C \subset N^-(v_k)$.*

Invariant 3. *For all $0 \leqslant i \leqslant n$, the partial order $\mathscr{P}_i$ has a linear extension that is a perfect factorizing permutation.*

Invariant 3 is equivalent to the following: whenever M is a module of the tournament, the members of $\mathscr{P}_i$ that intersect M are consecutive in the ordering on $\mathscr{P}_i$, and only the first and last of these can overlap with M.

The proof of the first two invariants is easy. Notice that, as G is a tournament, no element is lost when replacing C by $C \cap N^-(v_i)$, $\{v_i\}$, and $C \cap N^+(v_i)$. Let us prove the third invariant. Trivially $\mathscr{P}_0$ can be extended to a perfect factorizing permutation. Let us suppose that $\mathscr{P}_{i-1}$ also can. Let M be a module. Since $\mathscr{P}_i$ differs from $\mathscr{P}_{i-1}$ only in the class C that contains v_i, it is clear that our proof deals only with $C \in \mathscr{P}_{i-1}$. Let C_a be $C \cap N^-(v_i)$ and C_b be $C \cap N^+(v_i)$. There are three cases:

(1) If $C \subseteq M$, then the invariant is still true for M no matter how C is split.
(2) If $M \subset C$, then

- If $v_i \in M$, then M can overlap only the two classes C_a and C_b, and contains the class $\{v_i\}$ between them, so the invariant is true.
- If $v_i \notin M$ then, since M is a module, $M \subseteq C_a$ or $M \subseteq C_b$ depending on whether $M \subseteq N^+(v_i)$ or $M \subseteq N^-(v_i)$.

(3) If M overlaps C, then M intersects either the class on the left of C, or the class on its right. According to Invariant 2, this class is a singleton class, say $\{w\}$. Suppose without loss of generality that w is to the right of C. Then $C \subseteq N^-(w)$.

- If $v_i \notin M$, then $M \subset N^+(v_i)$ (because M is a module containing $w \in N^+(v_i)$). So $M \cap C = M \cap C_b$. Since C_b is located on the right of $\{v_i\}$, the invariant still holds.
- If $v_i \in M$ then $\forall x \in C_b$, $w \in N^+(x)$ and $v_i \in N^-(x)$. As $\{v_i, w\} \subset M$, $x \in M$ (it cannot cut a module), thus $C_b \subset M$. M can overlap only C_a, and contains $\{v_i\}$ and C_b: the invariant still holds.

Theorem 17. *Algorithm* 2 *computes a perfect factorizing permutation of any tournament in* $O(n + m)$ *time.*

Proof. The correctness is directly given by Invariants 1 and 3. At the n^{th} step, all classes are singleton, so the order is total. Since every vertex v_i is used once, and since computing C_a and C_b takes $O(n)$ time, the whole process takes $O(n^2)$ time, which is linear since G has $\Theta(n^2)$ arcs. □

5. Modular decomposition of directed graphs

If $G = (V, E)$ is a digraph. Let us define the following auxiliary objects:
The undirected graph $G_s = (V, E_s)$ such that $\{u, v\} \in E_s$ if and only if $(u, v) \in E$ *or* $(v, u) \in E$.
The undirected graph $G_d = (V, E_d)$, such that $\{u, v\} \in E_d$ if and only if $(u, v) \in E$ *and* $(v, u) \in E$.

Since undirected graphs are a special case of symmetric 2-structures, we may define the symmetric 2-structure $H(V, E_H) = G_s \wedge G_d$ (see Lemma 16). Let us assume that the colors of edges of H are indicated with the following labels:

- $E_H(u, v) = 0$ if $\{u, v\}$ is a non-edge ($\{u, v\}$ is a non-edge in both G_s and G_d).
- $E_H(u, v) = 1$ if $\{u, v\}$ is an edge ($\{u, v\}$ is an edge in both G_s and G_d).
- $E_H(u, v) = 2$ if (u, v) or (v, u) is a simple arc ($\{u, v\}$ is an edge in G_s but not in G_d).

Since the edges of G_d are a subset of the edges of G_s, there is no color for edges that are in G_d but not in G_s.

By Lemma 16, M is a module of H if and only if it is a module of both G_s and G_d.

Lemma 18. *Every module of G is a module of H.*

Proof. Suppose $X \subset V$ fails to be a module of G_d or G_s. Then there exist $x_1, x_2 \in X$ and $y \in V \backslash X$ such that, in G_d or G_s, y is a neighbor of x_1 but not of x_2. If this happens in G_s, then (x_1, y) or (y, x_1) is an arc of G, but neither (x_2, y) nor (y, x_2) is an arc of G, and X fails to be a module of G. If it happens in G_d, then both of (x_1, y) and (y, x_1) are arcs of G, but one of (x_2, y) and (y, x_2) fails to be an arc, and X again fails to be a module of G. □

Corollary 19. *There exists a way to order the children of each node in $T(H)$ so that the resulting leaf order is a factorizing permutation of G.*

Proof. Let X be a complete node of $T(H)$, and let $\mathscr{S} = \{S_1, S_2, \ldots, S_k\}$ be its children. Let $\mathscr{F}_X$ denote the tree-like family on domain $\mathscr{S}$ whose root is $\mathscr{S}$, whose leaves are $\{S_i | S_i \in \mathscr{S}\}$, and whose internal nodes are $\{\mathscr{S}' | \bigcup \mathscr{S}'$ is a strong module of $G\}$. Since no members of $\mathscr{F}_X$ overlap, it is a tree-like family on domain $\mathscr{S}$. If $\{S_1, S_2, \ldots, S_k\}$ is ordered according to their depth-first ordering of the inclusion tree of $\mathscr{F}_X$, all unions of children of X that are strong modules of G will be consecutive. By Lemma 18, every strong module of G is a node of $T(H)$ or a union of children of a complete node of $T(H)$, so applying such an ordering at every complete node of $T(H)$ will impose a factorizing permutation of G on the leaves. □

Our algorithm proceeds by ordering the children of nodes in $T(H)$ to obtain a factorizing permutation of G in linear time. Combining this with the algorithm of [2] gives a linear-time algorithm for modular decomposition of G.

Let X be a 0-complete or 1-complete node of $T(H)$, and let $S_1, S_2, \ldots, S_k$ be its children that are modules of G. Let R_X be the relation on $\{S_1, S_2, \ldots, S_k\}$ where $S_i R_X S_j$ iff $N^+(S_i) \cap (V \backslash X) = N^+(S_j) \cap (V \backslash X)$ and $N^-(S_i) \cap (V \backslash S) = N^-(S_j) \cap (V \backslash X)$. Clearly, R_X is an equivalence relation.

Lemma 20. *If X is a 0-complete or 1-complete node of $T(H)$ and Y is a strong module of G that is a union of children of X and not strong in H, then Y is the union of all the members of an equivalence class of R_X.*

Proof. Suppose X is 0-complete. Let $X_1 \ldots X_l$ be the children of X whose union is Y. In $G[X]$ there is no arc between X_i and X_j, for all $1 \leqslant i \neq j \leqslant l$. Y is therefore a parallel module of G, as $G[Y]$ is not connected. Furthermore each X_i is connected in G, else, each connected component of $G[X_i]$ would be a son of X, in T_H, instead of X_i. So X_i, as connected component of a parallel node, is a module of G. As Y is a module of G included in X, $X_i \subset Y$ and $X_j \subset Y$ are R_X-equivalent. Therefore, Y must be a union of children of X that are modules of G, members of a single R_X equivalence class. A son Z of X that is not included in Y is R_X-equivalent to X_i iff it is R_X-equivalent to X_j (as Y is a module). If this happen, then $Z \cup Y$ is a module of G. It is a parallel strong module (there is no arc between Y and Z) whose son Y also is a parallel strong module, contradiction. Y is therefore the union of a whole R_X equivalence class.

If X is 1-complete, then Y is a series module of G and of G_d, and the same proof holds, taking $\overline{G}$ instead of G. □

Lemma 21. *Let X be a 2-complete node of $T(H)$, let $S_1, S_2, \ldots, S_k$ be its children, and let $S = \{s_1, s_2, \ldots, s_k\}$ be arbitrary representatives from $S_1, S_2, \ldots, S_k$, respectively. $G[S]$ is a tournament. If $(s_1, s_2, \ldots, s_k)$ is a perfect factorizing permutation of $G[S]$, then every subfamily of $\{S_1, S_2, \ldots, S_k\}$ whose union is a module of G must be consecutive in $(S_1, S_2, \ldots, S_k)$.*

Proof. If M is a module of G that is a union of children of X, then $M \cap S$ is a module of $G[S]$. Since $(s_1, s_2, \ldots, s_k)$ is a perfect factorizing permutation of $G[S]$, $M \cap S$ is consecutive in this ordering, hence M is consecutive in $(S_1, S_2, \ldots, S_k)$. □

This gives an algorithm for finding the modular decomposition of G:

(1) Find G_s and G_d and H
(2) Find the modular decompositions $T(G_d)$ and $T(G_s)$ of G_d and G_s using one of the algorithms of [6,9,21].
(3) Find $T(H) = T(G_s) \wedge T(G_d)$
(4) At each 0-complete and 1-complete node X, order the children so that each equivalence class of R_X is consecutive.
(5) At each 2-complete node Y, select an arbitrary set S of representatives from the children. Order the children of Y according to a perfect factorizing permutation of $G[S]$
(6) The resulting leaf order of $T(H)$ is a factorizing permutation of G by Lemmas 20 and 21. Use the algorithm of [2] to find the modular decomposition of G.

Algorithm 3. Modular decomposition of a digraph

5.1. Complexity analysis

A linear time bound for Step 1 is trivial and the linear time bounds for Steps 2 and 6 are immediate from the cited results. At each node Y of Step 5, we may charge the cost of finding the perfect factorizing permutation of $G[S]$ to the corresponding edges of G at a cost of O(1) per edge, by Theorem 17. These edges all have Y as their least common ancestor in $T(H)$, so no edge of G is charged more than once. It remains to derive the linear time bounds for Steps 3 and 4.

5.1.1. Step 3

Theorem 22. *Let G be a digraph and $\mathscr{M}$ the set of its strong modules.*

$$\sum_{M \in \mathscr{M}} |M| \leqslant 2m + 3n$$

Furthermore if G is connected, then

$$\sum_{M \in \mathscr{M}} |M| \leqslant 2m + 2n$$

Proof. Let $\rho(G)$ be the sum $\sum_{M \in \mathscr{M}} |M|$. Clearly for a one-vertex digraph $\rho(G) = 1$. Let us suppose the theorem holds for digraphs up to $n-1$ vertices, and let G be a digraph having n vertices.

If G is not connected, then G has $k \geqslant 2$ connected components $G_1 \dots G_k$. Theorem 22 applies to each, so:

$$\forall\ 1 \leqslant i \leqslant k,\ \ \rho(G_i) \leqslant 2m_i + 2n_i.$$

Moreover, a strong module of G is either a strong module of $\{G_i\}$, $i \in \{1 \dots k\}$, or the vertex-set $V(G)$. Each arc of G appears in exactly one G_i, so that $\sum_i m_i = m$.

$$\begin{aligned}\rho(G) &= n + \rho(G_1) + \cdots + \rho(G_k)\\ &\leqslant n + 2m_1 + 2n_1 + \cdots + 2m_k + 2n_k\\ &\leqslant 2m + 3n.\end{aligned}$$

If G is connected, let G_i $1 \leqslant i \leqslant k$ be the maximal strong modules of G. Each one has less than n vertices, so

$$\forall\ 1 \leqslant i \leqslant k,\ \ \rho(G_i) \leqslant 2m_i + 3n_i.$$

There are two kinds of arcs in G: the $\sum m_i$ arcs that are internal to one G_i, and the $m' = m - \sum m_i$ "external" arcs joining two G_i's.

(1) If every G_i has at least two vertices, then every vertex of G is adjacent to at least two external arcs, so $m' \geqslant n$. $\sum m_i \leqslant m - n$.

$$\begin{aligned}\rho(G) &= n + \rho(G_1) + \cdots + \rho(G_k)\\ &\leqslant n + 2m_1 + 3n_1 + \cdots + 2m_k + 3n_k\\ &\leqslant 4n + 2\sum m_i\\ &\leqslant 2n + 2m.\end{aligned}$$

(2) Otherwise one G_i (say G_1) has only one vertex. G is connected and, if its internal arcs are removed, G remains connected. Therefore $m' \geqslant n-1$. $\sum m_i \leqslant m-n+1$

$$\begin{aligned}\rho(G) &= n + \rho(G_1) + \rho(G_2) + \cdots + \rho(G_k)\\ &\leqslant n+1+2m_2+3n_2+\cdots+2m_k+3n_k\\ &\leqslant n+1+2\sum m_i+3(n-1)\\ &\leqslant n+1+2m-2n+2+3n-3\\ &\leqslant 2n+2m. \qquad \square\end{aligned}$$

Corollary 23. *Step* 3 *takes* $O(n+m)$ *time.*

Proof. The sizes of G_d and G_s are $O(n+m)$, so the result follows immediately from Theorems 15 and 22. $\square$

5.1.2. Step 4

For Step 4, we must compute the equivalence classes of R_X at each 0-complete or 1-complete node X of $T(H)$. For this, it suffices to identify those members of the children $\{S_1, S_2, \ldots, S_k\}$ of X that are modules of G, and to group these according to their adjacencies with vertices in $V \backslash X$.

A solution to this problem on undirected graphs was first given in [26] and is a key step in the algorithms of [9,21]. Its generalization to a directed graph G using the decomposition tree T_H of H is straightforward, as we show next. We present here a variation of the computation method.

The algorithm is general to any inclusion tree T, not just T_H. Number the elements of V from 1 to n in the order of left-to-right appearance in an arbitrary embedding of T. This gives a factorizing permutation σ. Each node of T occupies a factor of σ. Let $le(X)$ be the first occurrence of a vertex of X in σ and $re(X)$ the last occurrence. If a node X of T has a cutter to its left in the ordering, let $lc(X)$ denote the number label of the leftmost of its cutters; otherwise let $lc(X) = le(X)$. If it has a cutter to its right, let $rc(X)$ denote the rightmost of its cutters; otherwise, let $rc(X) = re(X)$. The cutters are taken in G, not in H (where X is a module). Note that X is a module of G iff $lc(X) = le(X)$ and $rc(X) = re(X)$.

Lemma 24. *$le(X)$, $re(X)$, $lc(X)$ and $rc(X)$ can be computed, for all nodes X of T_H, in $O(n+m)$ time.*

Proof. Computing $re(X)$ and $le(X)$ can be done bottom-up. If X is a leaf, $lc(X) = rc(X) = \sigma(x)$. Else, $lc(X)$ and $rc(X)$ can be computed using the following recurrence relations, where $S_1, \ldots S_k$ are the children of X:

$$lc(X) = \min_{\sigma}(lc(S_1)\ldots lc(S_k), lc(re(S_1), le(S_2))\ldots lc(re(S_{k-1}), le(S_k)))$$

$$rc(X) = \max_{\sigma}(rc(S_1)\ldots rc(S_k), rc(re(S_1), le(S_2))\ldots rc(re(S_{k-1}), le(S_k))).$$

The proof for $lc(X)$ is that $lc(X) = \min_\sigma(rc(\{\sigma(i), \sigma(j)\}) \mid le(X) \leqslant i \neq j \leqslant re(X))$. As a vertex z cuts $\{\sigma(i), \sigma(j)\}$ then it cuts $\{\sigma(c), \sigma(c+1)\}, i \leqslant c < j$ (if two elements of an ordered set are different, two consecutive elements are differents. Here the ordered set is the edges between z and the factor $\sigma(i)\ldots\sigma(j)$). So we have : $lc(X) = \min_\sigma(rc(\{\sigma(i), \sigma(i+1)\}) \mid le(X) \leqslant i < re(X))$ Factorizing each S_i gives the result. Same proof for $rc(X)$.

A vertex x, preceded by y and trailed by z in σ, is used twice: one time for the computation of the node that is the least common ancestor of x and y, and one time for the least common ancestor of x and z. The key point of the complexity analysis is to show that $lc(\{x, y\})$ and $rc(\{x, y\})$ can be computed in $O(|N^+(x)| + |N^-(x)| + |N^+(y)| + |N^-(y)|)$ time.

To do this, bucket sort the edges of G according to σ, with vertex of origin as the primary key and destination vertex as the secondary key, in order to get $N^+(x)$ in sorted order of destination vertex at each vertex x. This takes $O(n+m)$ time [4]. Reverse the roles of primary and secondary key and sort again to get $N^-(x)$ in sorted order of vertex of origin in $O(n+m)$ time. $lc(\{x, y\})$ is the first vertex of $(N^+(x)\Delta N^+(y)) \cup (N^-(x)\Delta N^-(y))$. As these lists are sorted according to σ, it is easy to find. $rc(\{x, y\})$ is the last vertex of these lists. $\square$

For a node Y that is a module of G and child of a 0- or 1-complete node X of T_H, let $S^+(Y)$ be $N^+(Y)\backslash X$ and a $S^-(Y)$ be $N^-(Y)\backslash X$. $S^+(Y)$ and $S^-(Y)$ can be computed by taking any vertex $y \in Y$ and pruning elements of X from its adjacency list. Two children of X are R_X-equivalent iff they have the same lists. Since the lists are sorted according to σ, a partition refining algorithm using their first element, then the second, and so on until the last, separates the children of a 0- or 1-complete node X into R_X classes.

Lemma 25. *Step* 4 *takes* $O(n+m)$ *time.*

Proof. Lemma 24 give the time bound for discrimination of modules. That the lengths of the S^+ and S^- lists summed over all nodes of the tree is $O(m)$ follows easily from the fact that the sum $\sum_{x\in V}|S^+(\{x\})|+|S^-(\{x\})|$ of lengths of lists at the leaves is m, and at each internal node X, the lengths of its lists are at most half of the sum of lengths of the its children's lists. That is, if $\mathscr{C}(X)$ denotes the children of X, $|S^+(X)|\leqslant(1/2)\sum_{S\in\mathscr{C}(X)}|S^+(S)|$ and $|S^-(X)|\leqslant(1/2)\sum_{S\in\mathscr{C}(X)}|S^-(S)|$.

The algorithm is linear in the lengths of the S^+ and S^- lists, so it takes $O(n+m)$ time to find the R_X classes. □

References

[1] A. Brandstädt, V.B. Le, J. Spinrad, Graph Classes: a Survey, SIAM Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics, 1999.

[2] C. Capelle, M. Habib, F. de Montgolfier, Graph decomposition, factorizing permutations, Discrete Math. Theoret. Comput. Sci. 5 (1) (2002) http://dmtcs.loria.fr/volumes/abstracts/dm050104.abs.htm.

[3] M. Chein, M. Habib, M.C. Maurer, Partitive hypergraphs, Discrete Math. 37 (1981) 35–50.

[4] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2001.

[5] D.G. Corneil, H. Lerchs, L.K. Stewart, Complement reducible graphs, Discrete Appl. Math. 3 (1981) 163–174.

[6] A. Cournier, M. Habib, A new linear algorithm for modular decomposition, in: S. Tison (Ed.), Trees in algebra and programming—CAAP 94, 19th International Colloquium, Edinburgh, UK, Lecture Notes in Computer Science, vol. 787, Springer, Berlin, April 1994, pp. 68–84.

[7] E. Dahlhaus, Parallel algorithms for hierarchical clustering, and applications to split decomposition and parity graph recognition, J. Algorithms 36 (2) (2000) 205–240.

[8] E. Dahlhaus, J. Gustedt, R.M. McConnell, Efficient and practical algorithms for sequential modular decomposition, J. Algorithms 41 (2) (2001) 360–387.

[9] E. Dahlhaus, J. Gustedt, R.M. McConnell, Partially complemented representations of digraphs, Discrete Math. Theoret. Comput. Sci. 5 (1) (2002) 147–168.

[10] A. Ehrenfeucht, G. Rozenberg, Theory of 2-structures, Part I: clans, basic subclasses and morphisms, Theoret. Comput. Sci. 3 (70) (1990) 277–303.

[11] A. Ehrenfeucht, G. Rozenberg, Theory of 2-structures, Part II: Representations throught tree labelled families, Theoret. Comput. Sci. 3 (70) (1990) 304–342.

[12] A. Ehrenfreucht, H.N. Gabow, R.M. McConnell, S.J. Sullivan, An $O(n^2)$ divide-and-conquer algorithm for the prime Tree decomposition of 2-structures and the Modular Decomposition of graphs, J. Algorithms 16 (2) (1994) 283–294.

[13] T. Gallai, Transitiv orientierbare Graphen, Acta Math. Acad. Sci. Hungar. 18 (1967) 25–66.

[14] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.

[15] V. Giakoumakis, F. Roussel, H. Thuillier, On P_4-tidy graphs, Discrete Math. Theoret. Comput. Sci. 1 (1997) 17–41.

[16] M. Habib, Substitution des structures combinatoires, théorie et algorithmes, Thèse d''Etat, Université Pierre et Marie Curie (Paris VI), 1981.

[17] C.T. Hoàng, Perfect Graphs, Ph.D. Thesis, School of Computer Sciences, McGill University, Montreal, 1985.

[18] B. Jamison, S. Olariu, P_4-reducible graphs: a class of uniquely tree representable graphs, Stud. Appl. Math. 81 (1989) 79–87.

[19] R.M. McConnell, An $O(n^2)$ incremental algorithm for modular decomposition of graphs and 2-structures, Algorithmica 14 (1995) 209–227.

[20] R.M. McConnell, J. Spinrad, Linear-time modular decomposition and efficient transitive orientation of comparability graphs, in: Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, (Arlington, VA), ACM, New York, 1994, pp. 536–545.

[21] R.M. McConnell, J. Spinrad, Modular decomposition and transitive orientation, Discrete Math. 201 (1999) 189–241.

[22] R.H. Möhring, Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and boolean functions, Ann. Oper. Res. 6 (1985) 195–225.

[23] R.H. Möhring, F.J. Radermacher, Substitution decomposition for discrete structures and connections with combinatorial optimization, Ann. Discrete Math. 19 (1984) 257–356.

[24] H. Müller, Recognizing interval digraphs and interval bigraphs in polynomial time, Discrete Appl. Math. 78 (1997) 189–205.

[25] R. Paige, R.E. Tarjan, Three partition refinement algorithms, SIAM J. Comput. 16 (6) (1987) 973–989.

[26] J. Spinrad, P4-trees and substitution decomposition, Discrete Appl. Math. 39 (1992) 263–291.